

**EXP NO : 10**

**DATE : 23.04.25**

**GENERATE THREE ADDRESS CODES FOR A GIVEN EXPRESSION  
(ARITHMETIC EXPRESSION, FLOW OF CONTROL)**

**AIM:**

The aim is to generate Three-Address Code (TAC) for a given arithmetic expression and flow of control (e.g., if-else, loops). TAC is an intermediate representation used in compilers to simplify the task of code generation. It consists of simple instructions that make it easier to translate into machine-level code.

For example, for an arithmetic expression  $a = b + c * d$ , the TAC would break it down into simpler steps, using temporary variables to hold intermediate results.

**ALGORITHM:**

- The expression is read from the file using a file pointer
- Each string is read and the total no. of strings in the file is calculated.
- Each string is compared with an operator; if any operator is seen then the previous string and next string are concatenated and stored in a first temporary value and the three address code expression is printed
- Suppose if another operand is seen then the first temporary value is concatenated to the next string using the operator and the expression is printed.
- The final temporary value is replaced to the left operand value.

**PROGRAM:**

```
#include <stdio.h>
#include <string.h>
// Function to generate TAC for arithmetic expressions
void generateArithmeticTAC(const char* expr) {
    // Example: a = b + c * d
    // Expected TAC:
    // t1 = c * d
    // t2 = b + t1
    // a = t2

    char result[10], op1[10], op[2], op2[10];
    sscanf(expr, "%s = %s %s %s", result, op1, op, op2);

    if (strcmp(op, "+") == 0 || strcmp(op, "-") == 0) {
        printf("t1 = %s %s %s\n", op1, op, op2);
        printf("%s = t1\n", result);
    } else if (strcmp(op, "*") == 0 || strcmp(op, "/") == 0) {
        printf("t1 = %s %s %s\n", op1, op, op2);
        printf("%s = t1\n", result);
    } else {
        printf("Unsupported operation: %s\n", op);
    }
}
```

```
}
```

```
// Function to generate TAC for if-else statements
```

```
void generateIfElseTAC(const char* condition, const char* trueStmt, const char* falseStmt)
```

```
{
```

```
    // Example:
```

```
    // if (a < b) x = 1; else x = 2;
```

```
    // Expected TAC:
```

```
    // if a < b goto L1
```

```
    // goto L2
```

```
    // L1: x = 1
```

```
    // goto L3
```

```
    // L2: x = 2
```

```
    // L3:
```

```
    printf("if %s goto L1\n", condition);
```

```
    printf("goto L2\n");
```

```
    printf("L1: %s\n", trueStmt);
```

```
    printf("goto L3\n");
```

```
    printf("L2: %s\n", falseStmt);
```

```
    printf("L3:\n");
```

```
}
```

```
void generateWhileLoopTAC(const char* condition, const char* body) {
```

```
    // Example:
```

```
    // while (a < b) { x = x + 1; }
```

```
    // Expected TAC:
```

```
    // L1: if a >= b goto L2
```

```
    // x = x + 1
```

```
    // goto L1
```

```
    // L2:
```

```
    printf("L1: if %s goto L2\n", condition);
```

```
    printf("%s\n", body);
```

```
    printf("goto L1\n");
```

```
    printf("L2:\n");
```

```
}
```

```
int main() {
```

```
    // Example usage:
```

```
    printf("TAC for arithmetic expression:\n");
```

```
    generateArithmeticTAC("a = b + c");
```

```
    printf("\nTAC for if-else statement:\n");
```

```
    generateIfElseTAC("a < b", "x = 1", "x = 2");
```

```
    printf("\nTAC for while loop:\n");
```

```
    generateWhileLoopTAC("a >= b", "x = x + 1");
```

```
    return 0;
```

```
}
```

## OUTPUT :

▲ TAC for arithmetic expression:

```
t1 = b + c
```

```
a = t1
```

TAC for if-else statement:

```
if a < b goto L1
```

```
goto L2
```

```
L1: x = 1
```

```
goto L3
```

```
L2: x = 2
```

```
L3:
```

TAC for while loop:

```
L1: if a >= b goto L2
```

```
x = x + 1
```

```
goto L1
```

```
L2:
```

## RESULT:

Thus the above program is the simplified example and a complete implementation and it would need to handle more complex expressions, nested control structures, and ensure proper parsing of the input.